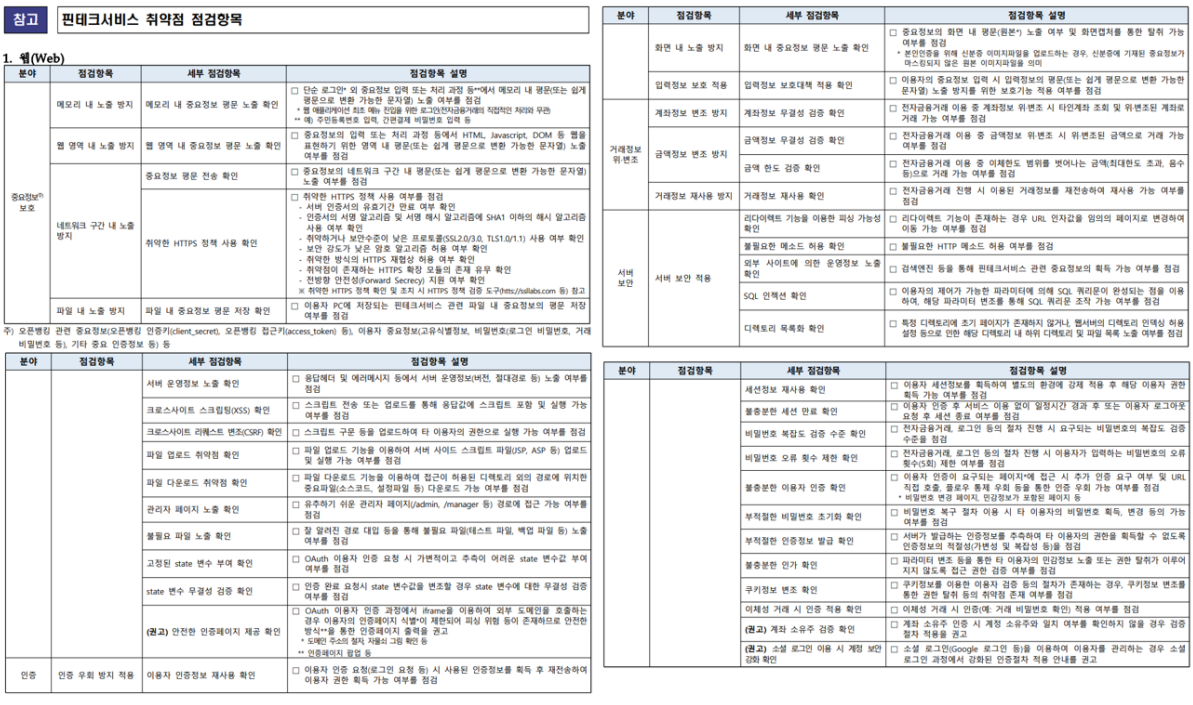
프로젝트

- 핀테크 금융보안원 가이드에 따라서 보안 솔루션 구축 프로젝트



참고 핀테크서비스 취약점 점검항목

1. 웹(Web)

분야	점검항목	세부 점검항목	점검항목 설명
중요정보[주] 보호	메모리 내 노출 방지	메모리 내 중요정보 평문 노출 확인	□ 단순 로그인* 외 중요정보 입력 또는 처리 과정 등**에서 메모리 내 평문(또는 쉽게 평문으로 변환 가능한 문자열) 노출 여부를 점검 * 웹 애플리케이션 최초 메뉴 진입을 위한 로그인(전자금융거래의 직접적인 처리와 무관) ** 예) 주민등록번호 입력, 간편결제 비밀번호 입력 등
	웹 영역 내 노출 방지	웹 영역 내 중요정보 평문 노출 확인	□ 중요정보의 입력 또는 처리 과정 등에서 HTML, Javascript, DOM 등 웹을 표현하기 위한 영역 내 평문(또는 쉽게 평문으로 변환 가능한 문자열) 노출 여부를 점검
	네트워크 구간 내 노출 방지	중요정보 평문 전송 확인	□ 중요정보의 네트워크 구간 내 평문(또는 쉽게 평문으로 변환 가능한 문자열) 노출 여부를 점검
		취약한 HTTPS 정책 사용 확인	□ 취약한 HTTPS 정책 사용 여부를 점검 - 서버 인증서의 유효기간 만료 여부 확인 - 인증서의 서명 알고리즘 및 서명 해시 알고리즘에 SHA1 이하의 해시 알고리즘 사용 여부 확인 - 취약하거나 보안수준이 낮은 프로토콜(SSL2.0/3.0, TLS1.0/1.1) 사용 여부 확인 - 보안 강도가 낮은 암호 알고리즘 허용 여부 확인 - 취약한 방식의 HTTPS 재협상 허용 여부 확인 - 취약점이 존재하는 HTTPS 확장 모듈의 존재 유무 확인 - 전방향 안전성(Forward Secrecy) 지원 여부 확인 ※ 취약한 HTTPS 정책 확인 및 조치 시 HTTPS 정책 검증 도구(https://ssllabs.com 등) 참고
	파일 내 노출 방지	파일 내 중요정보 평문 저장 확인	□ 이용자 PC에 저장되는 핀테크서비스 관련 파일 내 중요정보의 평문 저장 여부를 점검

주) 오픈뱅킹 관련 중요정보(오픈뱅킹 인증키(client_secret), 오픈뱅킹 접근키(access_token) 등), 이용자 중요정보(고유식별정보, 비밀번호(로그인 비밀번호, 거래 비밀번호 등), 기타 중요 인증정보 등) 등

분야	점검항목	세부 점검항목	점검항목 설명
		서버 운영정보 노출 확인	□ 응답헤더 및 에러메시지 등에서 서버 운영정보(버전, 절대경로 등) 노출 여부를 점검
		크로스사이트 스크립팅(XSS) 확인	□ 스크립트 전송 또는 업로드를 통해 응답값에 스크립트 포함 및 실행 가능 여부를 점검
		크로스사이트 리퀘스트 변조(CSRF) 확인	□ 스크립트 구문 등을 업로드하여 타 이용자의 권한으로 실행 가능 여부를 점검
		파일 업로드 취약점 확인	□ 파일 업로드 기능을 이용하여 서버 사이드 스크립트 파일(JSP, ASP 등) 업로드 및 실행 가능 여부를 점검
		파일 다운로드 취약점 확인	□ 파일 다운로드 기능을 이용하여 접근이 허용된 디렉토리 외의 경로에 위치한 중요파일(소스코드, 설정파일 등) 다운로드 가능 여부를 점검
		관리자 페이지 노출 확인	□ 유추하기 쉬운 관리자 페이지(/admin, /manager 등) 경로에 접근 가능 여부를 점검
		불필요 파일 노출 확인	□ 잘 알려진 경로 대입 등을 통해 불필요 파일(테스트 파일, 백업 파일 등) 노출 여부를 점검
		고정된 state 변수 부여 확인	□ OAuth 이용자 인증 요청 시 가변적이고 추측이 어려운 state 변수값 부여 여부를 점검
		state 변수 무결성 검증 확인	□ 인증 완료 요청시 state 변수값을 변조할 경우 state 변수에 대한 무결성 검증 여부를 점검
		(권고) 안전한 인증페이지 제공 확인	□ OAuth 이용자 인증 과정에서 iframe을 이용하여 외부 도메인을 호출하는 경우 이용자의 인증페이지 식별*이 제한되어 피싱 위협 등이 존재하므로 안전한 방식**을 통한 인증페이지 출력을 권고 * 도메인 주소의 철자 자물쇠 그림 확인 등 ** 인증페이지 팝업 등
인증	인증 우회 방지 적용	이용자 인증정보 재사용 확인	□ 이용자 인증 요청(로그인 요청 등) 시 사용된 인증정보를 획득 후 재전송하여 이용자 권한 획득 가능 여부를 점검

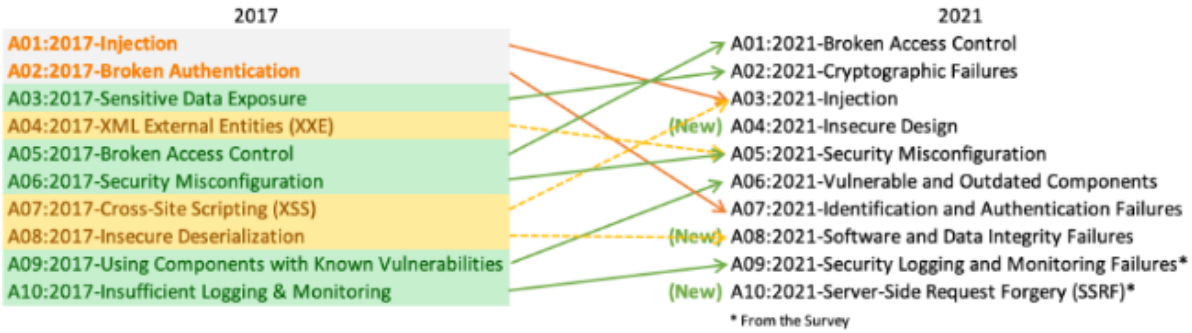
분야	점검항목	세부 점검항목	점검항목 설명
	화면 내 노출 방지	화면 내 중요정보 평문 노출 확인	□ 중요정보의 화면 내 평문(원본) 노출 여부 및 화면캡처를 통한 탈취 가능 여부를 점검 * 본인인증을 위해 신분증 이미지파일을 업로드하는 경우, 신분증에 기재된 중요정보가 마스킹되지 않은 원본 이미지파일을 의미
	입력정보 보호 적용	입력정보 보호대책 적용 확인	□ 이용자의 중요정보 입력 시 입력정보의 평문(또는 쉽게 평문으로 변환 가능한 문자열) 노출 방지를 위한 보호기능 적용 여부를 점검
거래정보 위변조	계좌정보 변조 방지	계좌정보 무결성 검증 확인	□ 전자금융거래 이용 중 계좌정보 위변조 시 타인계좌 조회 및 위변조된 계좌로 거래 가능 여부를 점검
	금액정보 변조 방지	금액정보 무결성 검증 확인	□ 전자금융거래 이용 중 금액정보 위변조 시 위변조된 금액으로 거래 가능 여부를 점검
		금액 한도 검증 확인	□ 전자금융거래 이용 중 이체한도 범위를 벗어나는 금액(최대한도 초과, 음수 등)으로 거래 가능 여부를 점검
	거래정보 재사용 방지	거래정보 재사용 확인	□ 전자금융거래 전행 시 이용된 거래정보를 재전송하여 재사용 가능 여부를 점검
서버 보안	서버 보안 적용	리다이렉트 기능을 이용한 피싱 가능성 확인	□ 리다이렉트 기능이 존재하는 경우 URL 인자값을 임의의 페이지로 변경하여 이동 가능 여부를 점검
		불필요한 메소드 허용 확인	□ 불필요한 HTTP 메소드 허용 여부를 점검
		외부 사이트에 의한 운영정보 노출 확인	□ 검색엔진 등을 통해 핀테크서비스 관련 중요정보의 획득 가능 여부를 점검
		SQL 인젝션 확인	□ 이용자의 제어가 가능한 파라미터에 의해 SQL 쿼리문이 완성되는 점을 이용하여, 해당 파라미터 변조를 통해 SQL 쿼리문 조작 가능 여부를 점검
		디렉토리 목록화 확인	□ 특정 디렉토리에 초기 페이지가 존재하지 않거나, 웹서버의 디렉토리 인덱싱 허용 설정 등으로 인한 해당 디렉토리 내 하위 디렉토리 및 파일 목록 노출 여부를 점검

분야	점검항목	세부 점검항목	점검항목 설명
		세션정보 재사용 확인	□ 이용자 세션정보를 획득하여 별도의 환경에 강제 적용 후 해당 이용자 권한 획득 가능 여부를 점검
		불충분한 세션 만료 확인	□ 이용자 인증 후 서비스 이용 없이 일정시간 경과 후 또는 이용자 로그아웃 요청 후 세션 종료 여부를 점검
		비밀번호 복잡도 검증 수준 확인	□ 전자금융거래, 로그인 등의 절차 전행 시 요구되는 비밀번호의 복잡도 검증 수준을 점검
		비밀번호 오류 횟수 제한 확인	□ 전자금융거래, 로그인 등의 절차 전행 시 이용자가 입력하는 비밀번호의 오류 횟수(5회) 제한 여부를 점검
		불충분한 이용자 인증 확인	□ 이용자 인증이 요구되는 페이지에 접근 시 추가 인증 요구 여부 및 URL 직접 호출, 플로우 통제 우회 등을 통한 인증 우회 가능 여부를 점검 * 비밀번호 변경 페이지, 민감정보가 포함된 페이지 등
		부적절한 비밀번호 초기화 확인	□ 비밀번호 복구 절차 이용 시 타 이용자의 비밀번호 획득, 변경 등의 가능 여부를 점검
		부적절한 인증정보 발급 확인	□ 서버가 발급하는 인증정보를 추측하여 타 이용자의 권한을 획득할 수 없도록 인증정보의 적절성(가변성 및 복잡성 등)을 점검
		불충분한 인가 확인	□ 파라미터 변조 등을 통한 타 이용자의 민감정보 노출 또는 권한 탈취가 이루어지지 않도록 접근 권한 검증 여부를 점검
		쿠키정보 변조 확인	□ 쿠키정보를 이용한 이용자 검증 등의 절차가 존재하는 경우, 쿠키정보 변조를 통한 권한 탈취 등의 취약점 존재 여부를 점검
		이체성 거래 시 인증 적용 확인	□ 이체성 거래 시 인증(예: 거래 비밀번호 확인) 적용 여부를 점검
		(권고) 계좌 소유주 검증 확인	□ 계좌 소유주 인증 시 계정 소유주와 일치 여부를 확인하지 않을 경우 검증 절차 적용을 권고
		(권고) 소셜 로그인 이용 시 계정 보안 강화 확인	□ 소셜 로그인(Google 로그인 등)을 이용하여 이용자를 관리하는 경우 소셜 로그인 과정에서 강화된 인증절차 적용 안내를 권고

- 점검 항목 가이드에 따라서 보안 솔루션 코딩

OWASP TOP 10

- 2021년 9월 24일 기준



A01: Broken Access Control	취약한 접근 제어 : 권한/인가
A02: Cryptographic Failures	암호화 실패
A03: Injection	인젝션
A04: Insecure Design	안전하지 않은 설계
A05: Security Misconfiguration	보안 설정 오류
A06: Vulnerable and Outdated Components	취약하고 지원이 종료된 구성 요소
A07: Identification and Authentication Failures	식별 및 인증 실패
A08: Software and Data Integrity Failures	소프트웨어 및 데이터 무결성 오류
A09: Security Logging and Monitoring Failures	보안 로깅 및 모니터링 오류
A10: Server-Side Request Forgery	SSRF, 서버 측 요청 변조

A01: Broken Access Control(취약한 접근 제어 : 권한/인가)

- 접근 제어(Access control)는 사용자가 권한을 벗어난 행동을 할 수 없게 정책을 만들고 실행하는 기능이다.
- 접근 제어가 취약하게 구현이 되면 사용자는 권한을 벗어나서 인가되지 않은 데이터에 접근해서, 조작이나 삭제를 하여 무결성을 침해할 수 있다.

💡 **취약점 예시**

1. 특정 사용자에게 부여해야 하는 권한을 모든 사용자에게 부여하는 경우
2. 인증되지 않은 사용자가 인증이 필요한 페이지를 강제로 탐색 가능한 경우
3. 사용자로 로그인해서 관리자 권한으로 활동할 수 있는 경우
4. POST, PUT, DELETE API 요청에 대해 접근 제어가 누락된 경우

💡 **예방방법**

1. 공용 리소스를 제외하고 기본적으로 접근을 거부하는 정책 수립
2. 접근제어 정책이 애플리케이션 전체에 일괄 적용되도록 확인

애플리케이션이 계정 정보를 조회하는 SQL을 호출하는 경우, 이를 조작해서 허가 되지 않은 데이터에 접근

```
pstmt.setString(1, request.getParameter("userId"));
ResultSet results = pstmt.executeQuery( );
```

```
GET /accountInfo?userId=77 HTTP/1.1
```

공격자가 HTTP 요청을 할 때 'userID' 파라미터를 조작해서 타 사용자의 계정 정보를 요청할 수 있다.

서버 측에서 'userID'를 검증하지 않는다면 공격자는 모든 사용자의 계정 정보에 접근 가능

관리자 페이지 접근제어

> 스캐닝 방법 : 요청에 대한 접근제어 및 나뉘어 있는 DB 사용자 그룹에 대해서 확인

도메인에 list에 있는 인자들을 입력 받아서 url이 있는지 확인 후
-HTTP, URL 에러가 발생하면 출력x
-만약 예외 처리사항이 없다면 y+=1을 하여 취약하다고 출력 그렇지 않다면 양호

admin_check.py 1

owasp top10 > admin_check.py > ...

```python
from urllib.request import *
from urllib.error import *

list = ['admin', 'administrator', 'mastser', 'manager', 'management'
        'system', 'test', 'anonymous']

x = input('도메인 주소를 입력해주세요(ex>www.test.com) : ')

y = 0

for k in list:
    url = 'http://' + x + '/' + k
    try:
        request = urlopen(url)
    except HTTPError as e:
        continue
    except URLError as e:
        continue
    else:
        y+=1
        print('\n[',i,']',url)

if y>=1:
    print('\n관리자 페이지 노출 취약점 : 취약\n')
else:
    print('\n관리자 페이지 노출 취약점 : 양호\n')
```

A02: Cryptographic Failures(암호화 실패)

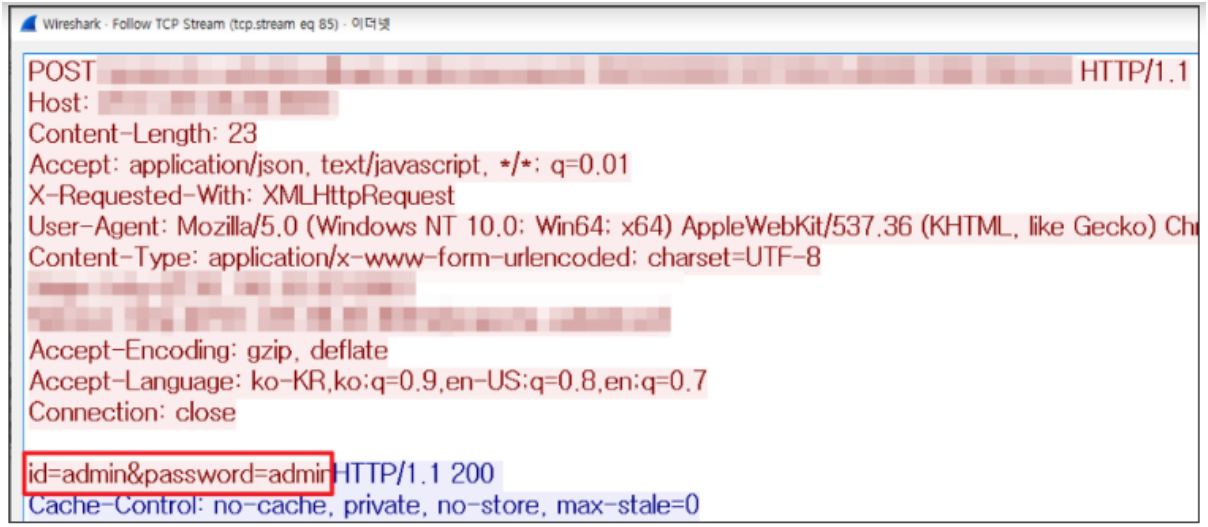
- '민감 데이터 노출 → 암호화 실패' 로 명칭 변경
- 암호화에 오류가 있거나 미흡하면 민감 데이터가 노출되어 기밀성에 취약해진다.
- 특히 개인정보와 금융데이터 같은 경우 강력하게 영향을 받는다.

취약점 예시

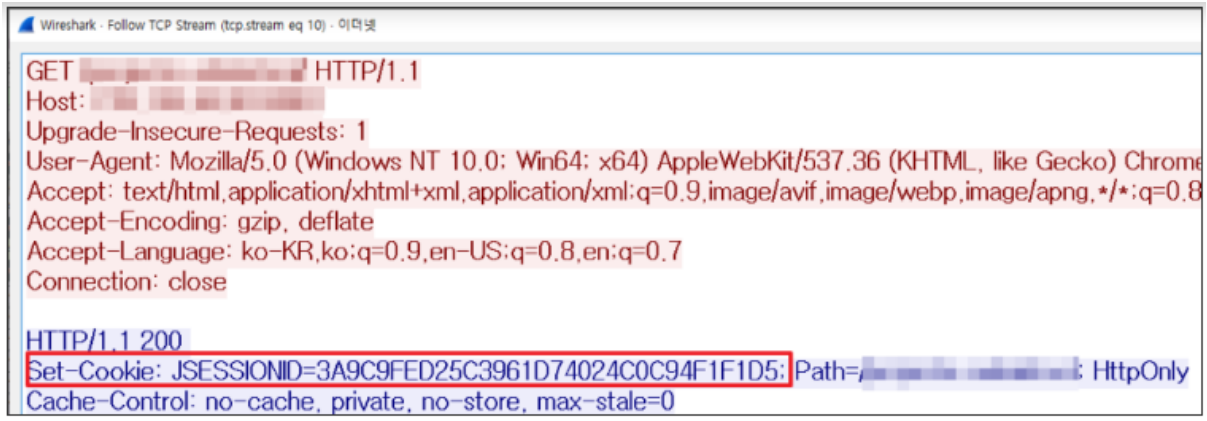
1. 내외부망에 관계없이 데이터가 전송 구간에서 평문으로 전송되는 경우(HTTP, FTP, TELNET 등)
2. 취약한 암호화 프로토콜 이용하는 경우
3. 취약한 암호화 알고리즘 사용하는 경우
4. 취약한 암호화 컴포넌트를 사용하는 경우
5. 고정된 암호문 사용하는 경우
6. 사설 인증서 사용
7. 암호키 관리 미흡한 경우

예방 방법

1. FTP, TELNET과 같은 레거시 프로토콜 미사용
2. 최신 버전의 암호 프로토콜 및 안전한 암호 알고리즘 사용
3. HSTS(HTTP를 HTTPS로 강제 리다이렉트) 설정
4. 암호화 시 암호문이 고정되지 않도록 의사 난수 생성기를 포함
5. 신뢰할 수 있는 기관에서 발급한 인증서 사용

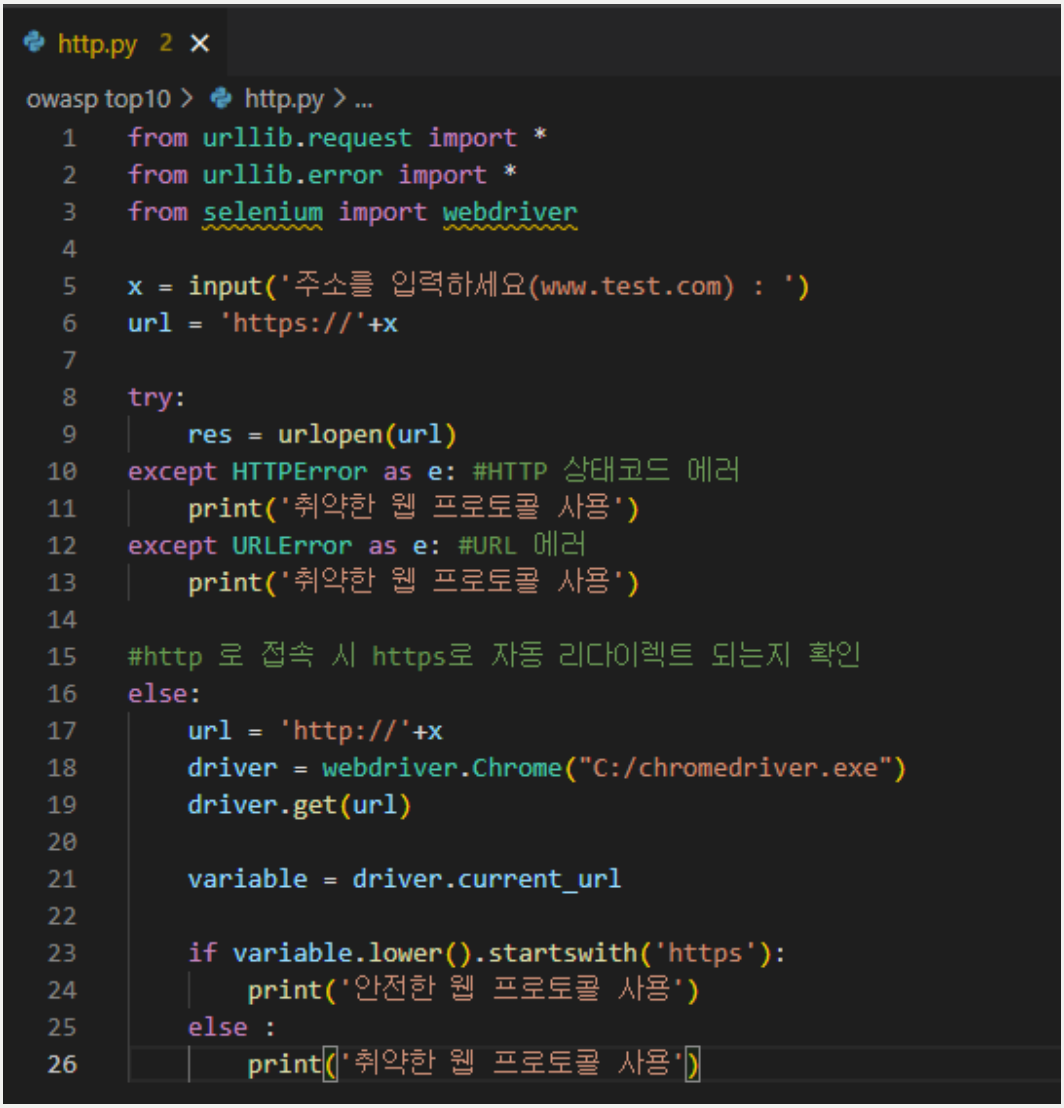


공격자가 스니핑하고 있는 네트워크에서 중요 정보가 포함된 데이터를 평문으로 전송하는 경우, 공격자는 평문으로 노출된 아이디와 패스워드를 획득해 해당 시스템에 로그인 가능(아이디와 패스워드 노출로 세션 ID 획득 후 시스템에 접근 가능)



> 스캐닝 방법 : 인증서 기관 확인하여 신뢰할 수 있는 곳인지 아닌지 판단, 암호 알고리즘 버전 확인, 레거시 프로토콜 사용 확인(미흡/양호로 결과 출력)

http가 취약하므로 https 접속 후 리다이렉트 기능 포함하는 지 확인



http.py 2

owasp top10 > http.py > ...

```python
from urllib.request import *
from urllib.error import *
from selenium import webdriver

x = input('주소를 입력하세요(www.test.com) : ')
url = 'https://'+x

try:
    res = urlopen(url)
except HTTPError as e: #HTTP 상태코드 에러
    print('취약한 웹 프로토콜 사용')
except URLError as e: #URL 에러
    print('취약한 웹 프로토콜 사용')

#http 로 접속 시 https로 자동 리다이렉트 되는지 확인
else:
    url = 'http://'+x
    driver = webdriver.Chrome("C:/chromedriver.exe")
    driver.get(url)

    variable = driver.current_url

    if variable.lower().startswith('https'):
        print('안전한 웹 프로토콜 사용')
    else :
        print('취약한 웹 프로토콜 사용')
```

A03: Injection(인젝션)

- XSS가 인젝션 항목에 포함
- 전달 되는 인수값(데이터: 파라미터, 헤더, URL, 쿠키 등)을 조작해서 서버 측에서 명령어나 쿼리문을 인식하게 해서 발생하게 하는 취약점

💡 **취약점 예시**

1. SQL Injection
2. OS Command Injection
3. ORM Injection
4. Cross-site Scripting

💡 **예방 방법**

사용자 입력이 SQL 문법으로 인식되지 않도록 Binding 변수 사용

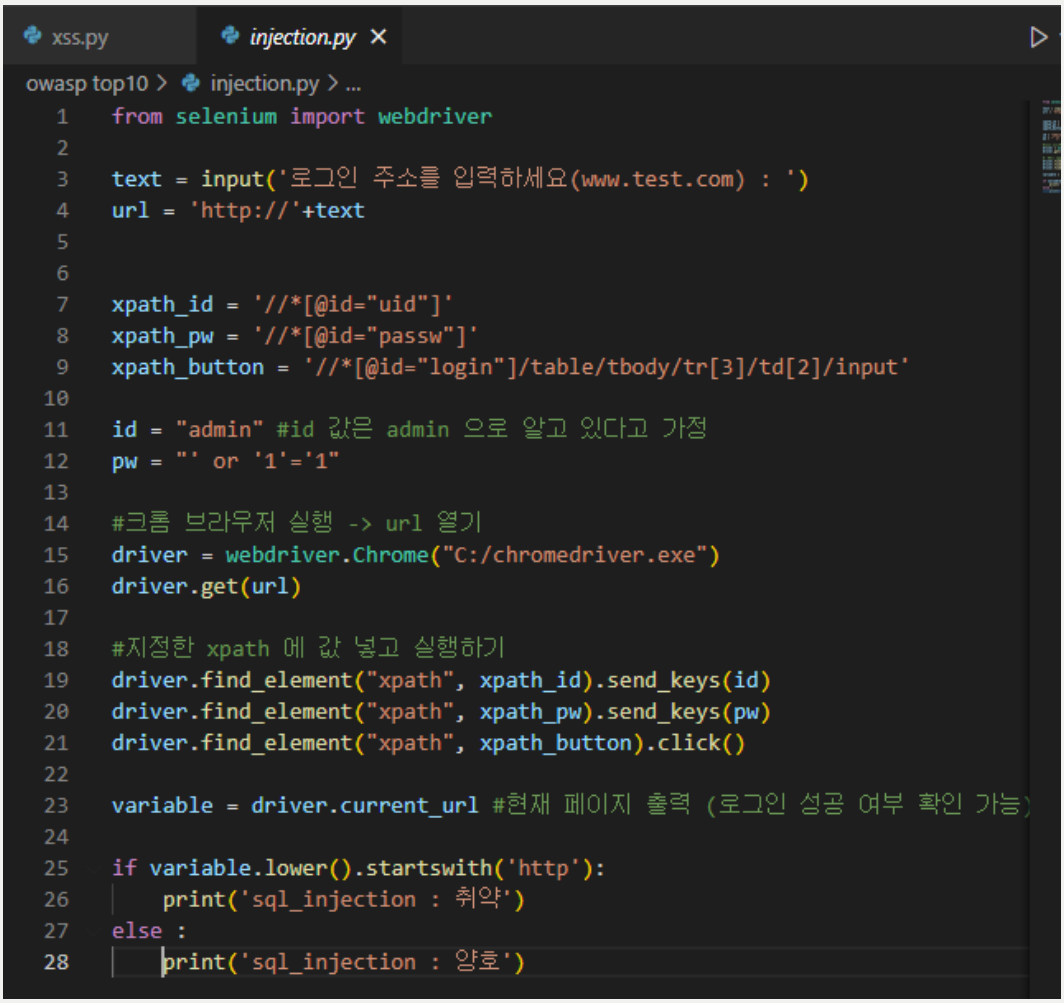
id 파라미터 조작하여 해당 컬럼 모든 데이터 조회

```
// SQL 호출을 취약하게 구현한 애플리케이션
\
String query = "SELECT \* FROM accounts WHERE custID='" + request.getParameter("id")
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getPar
```

```
http://example.com/app/accountView?id=' or '1'='1
```

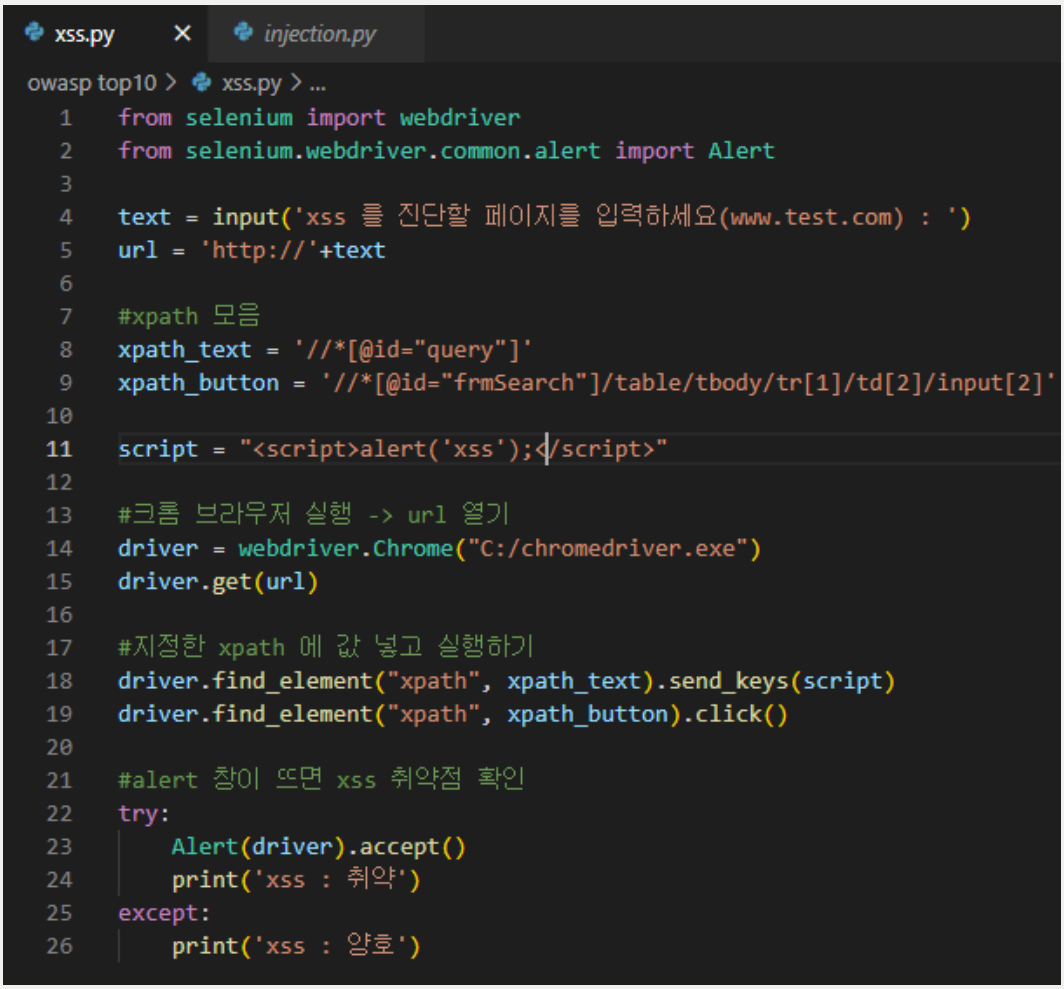
> 스캐닝 방법 : GET방식이라면 여러 특수 문자를 넣는 코드를 코딩해서 database Error 발생여부 체크, POST방식이라면 Body 값에 특수문자 삽입이 가능한지 체크하여 injection 취약점을 스캐닝 한다.

- DB와 연동되어 있는 입력창에 SQL injection 문을 삽입해 출력이 되는지 확인하는 취약점 확인



XSS

-악성 슼크립트를 확인하기 위해 alert창을 실행하여 xpath로 따온 경로가 xss에 취약한 페이지라면 xss alert 발생



A04: Insecure Design(안전하지 않은 설계)

- 보안 결함을 의미한다.

💡 1. 요구사항 및 리소스 관리: 모든 데이터 자산과 예상되는 비즈니스 로직의 기밀성, 무결성 가용성 및 신뢰성에 관한 보안 요구사항을 포함하여 기획
2. 보안 설계: 알려진 공격 기법을 방지하기 위해 위협을 지속적으로 평가하고 코드가 안전하게 설계되고 테스트됐는지 확인하는 문화 및 방법론 적용
3. 보안 개발 생명 주기: 전체 프로젝트 및 소프트웨어 유지 관리의 전반에 걸쳐 프로젝트 시작 단계에서부터 보안 전문가의 검증 필요

> 4번 항목은 개발 후 코드를 수정하여 유지 보수하여 방어해야 한다.**<구현불가>**

A05: Security Misconfiguration (보안 설정 오류)

- 애플리케이션 업데이트할 때 보안성을 고려하여 설정을 제대로 하여 취약성을 보수한다.

💡 **취약점 예시**
1. 불필요 기능 활성화 및 설치
2. 관리자 계정 변경 하지 않고 사용
3. 보안 헤더 설정 누락
4. 에러 페이지에 에러 정보 노출
5. 서버 단 보안 설정 누락

💡 **예방 방법**
1. 애플리케이션 설치 시 불필요한 기능, 구성요소, 샘플, 문서 등 제거
2. 보안 헤더 설정
3. 모든 환경에 보안 설정 검증

> 스캐닝 방법 : 관리자 계정 초기 아이디, 패스워드를 입력하여 로그인이 되면 취약성을 표시하고 안된다면 양호하다고 표시

- 요청 응답 메시지에 특정 패턴 확인
-사용자 지정하지 않은 에러 페이지에 그대로 내부 정보 노출 되어있기에 해당한다면 보안설정 오류

errorpage_check.py

owasp top10 > errorpage_check.py > ...

```python
from selenium import webdriver
from urllib.request import *
from urllib.error import *

#xpath 를 통한 오류구문 리스트 (xpath 에 입력시에 발생하는 오류구문)
error_list1 = ['No results were found for the query', 'syntax error', '로변환하지 못했습\
니다','변환하는 중 구문 오류가 발생했습니다.','따옴표가 짝이 맞지 않습니다.','You have \
an error in your SQL syntax','Unclosed quotation mark after the characte string','Orac\le Text error:']

text = input('에러페이지 확인 주소 입력(www.test.com) : ')
url = 'http://'+text

#xpath
xpath_text = '//*[@id="query"]'
xpath_button = '//*[@id="frmSearch"]/table/tbody/tr[1]/td[2]/input[2]'

script = "'"

driver = webdriver.Chrome("C:/chromedriver.exe")
driver.get(url)

#xpath 값 넣고 실행
driver.find_element("xpath", xpath_text).send_keys(script)
driver.find_element("xpath", xpath_button).click()

html = driver.page_source #현재 페이지 html 소스코드 가져오기

summ = 0
#html 에 오류구문이 있는지 검사
for i in error_list1:
    if i in html:
        summ += 1
```

```python
if summ != 0: #html 에서 오류구문 발견
    print('에러페이지 정보노출 : 취약')
else: #html 에서 오류구문 미발견
    error_list2 = ['Apache', 'nginx', 'IIS'] #url 직접 입력을 통한 오류구문 리스트
    url = url+'/errorpage_check'

    #크롬 브라우저 실행 -> url 열기
    driver = webdriver.Chrome("C:/chromedriver.exe")
    driver.get(url)
    try:
        res = urlopen(url) # url 열기
    except HTTPError as e: # HTTP 에러발생 -> 웹서버 정보 나오는지 검증
        summ = 0
        html = driver.page_source
    #html 에 웹서버 정보가 나오는지 검사
        for j in error_list2:
            if j in html:
                print('에러페이지 정보노출 : 취약')
                break
    else:
        summ += 1
        if summ == int(len(error_list2)):
            print('에러페이지 정보노출 : 양호')
```

A06: Vulnerable and Outdated Components(취약하고 지원이 종료된 구성 요소)

- 취약 버전 소프트웨어 사용하는 경우

💡 **취약점 예시**

1. 지원 종료된 OS 사용
2. 취약점이 존재하는 애플리케이션 사용
3. 취약점이 존재하는 프레임 워크 사용
4. 알려진 취약점 존재하는 버전 라이브러리 사용

💡 **예방 방법**

1. 불필요 소프트웨어 제거
2. 소프트웨어 버전 체크
3. 패치 관리 프로세스를 통해 소프트웨어 최신 버전 유지
4. 알려진 취약점 지속적으로 모니터링해 소프트웨어 사용 확인

> 스캐닝 버전: 오래된 소프트웨어 버전 체크, 패치 버전 체크, 오랫동안 (기간에 맞게) 사용하지 않은 라이브러리 체크하여 취약점 양호 미흡 체크

-웹 페이지에서 사용된 각 프레임워크, 라이브러리, 패치 정보, 종료된 서비스 취약점 확인

CVE.py

owasp top10 > CVE.py > ...

```python
from selenium import webdriver #크롤링을 위한 모듈
from bs4 import BeautifulSoup

text = 'https://cve.mitre.org/cve/search_cve_list.html'

xpath_search = '//*[@id="CenterPane"]/form/div[1]/input'
xpath_button = '//*[@id="CenterPane"]/form/div[2]/input'

search = input('점검할 서비스 입력 ( ex : ubuntu 20.04 ) : ')

driver = webdriver.Chrome("C:/chromedriver.exe")
driver.get(text)

#xpath 값 넣고 실행
driver.find_element("xpath", xpath_search).send_keys(search)
driver.find_element("xpath", xpath_button).click()

variable = driver.current_url #현재 페이지 출력 (로그인 성공 여부 확인 가능)

html = driver.page_source
soup = BeautifulSoup(html,'html.parser')
td_tag = soup.find_all('td') #td 태그 다 가져오기

count = 0
for i in td_tag:
    count+=1

if int(count)>=21:
    print('버전 업데이트 또는 변경이 필요합니다.')
else:
    print('안전합니다')
```

A07: Identification and Authentication Failures(식별 및 인증 실패)

- 취약한 인증 및 식별을 실패

💡 **취약점 예시**

1. 인증 실패에 대한 제한이 없어 Brute forcing 공격에 노출되는 경우
2. URL에 인증 세션 ID가 노출되는 경우(GET Method)
3. 세션 타임아웃이 없거나 로그아웃 후 세션 파기를 하지 않는 경우

💡 **예방 방법**

1. 안전한 비밀번호 생성 정책 및 인증 실패 횟수 제한 적용
2. 로그인 시 새로운 세션 ID를 생성하고 인증 세션은 암호화된 채널에서 헤더를 통해 전송
3. 세션 파기 및 만료 정책 수립

> 스캐닝 방법 : URL 인증 세션노출 확인하여 ID값 보이는지 확인, 무반응 세션에 대한 타임아웃 체크

-아이디, 패스워드, 모든 조합 대입
-로그인 여부 확인

bruteforce.py

```python
from itertools import product
from selenium import webdriver
from urllib.error import *
from urllib.request import *
import sys

#brutefore 공격에 사용할 문자
words ='wxyz'

#xpath 모음
xpath_id = '//[@id="user_login"]'
xpath_pw = '//[@id="user_pass"]'
xpath_click = '//*[@id="wp-submit"]'

text = input('로그인 주소를 입력하세요(www.test.com) : ')
id = input('ID : ')

url = 'http://' + text
error = 0
for passwd_length in range(1,3): #자릿수 지정 (1~2)
    admin_passwd = product(words, repeat=passwd_length) #words 안의 문자를 하나씩 끊어서 경우의 수 만들기

    #bruteforce 공격 실행
    for passwd_tmp in admin_passwd:
        passwd = ''
        passwd = ''.join(passwd_tmp)
        #크롬 브라우저 실행 -> url 열기
        driver = webdriver.Chrome("C:/chromedriver.exe")
        driver.get(url)
        #지정한 xpath 에 값 넣고 실행하기
        driver.find_element('xpath',xpath_id).send_keys(id)
        driver.find_element('xpath',xpath_pw).send_keys(passwd)
        driver.find_element('xpath',xpath_click).click()
        try:
            res = urlopen(driver.current_url)
            html = driver.page_source
        except HTTPError as e: # HTTP 에러발생 -> 무시하고 진행
            continue
        else: #로그인 성공시 뜨는 페이지 특징으로 bruteforce 공격 성공 여부 확인
            if "안녕하세요" in html:
                print('Brute Force : 취약')
                error = 1
                sys.exit()
            else:
                continue
if error != 1:
    print('Brute Force : 양호')
```

A08: Software and Data Integrity Failures(소프트웨어 및 데이터 무결성 오류)

- 신뢰할 수 없는 소스, 저장소 및 CDN, 플러그인, 라이브러리, 모듈에 의존하는 경우에 발생
- 안전하지 않은 애플리케이션이 변조 되면서 무결성이 훼손

💡 **취약점 예시**
1. 무결성 검증이 없어 변조가 가능한 경우
2. 업데이트 공급망에 대한 검증이 없는 경우
3. 직렬화된 데이터에 대한 무결성 검증이 없는 경우

💡 **예방 방법**
1. 전자서명, 해시 알고리즘 등을 사용해 애플리케이션 무결성을 검증
2. 사용하고 있는 라이브러리가 신뢰할 수 있는 저장소를 사용하고 있는지 확인하고, 중요한 서비스라면 내부 저장소를 별도로 지정해 사용

> 서버와 클라이언트 간 내부의 통신에 해쉬 값 이동 확인 불가하므로 **<구현불가>**

A09: Security Logging and Monitoring Failures(보안 로깅 및 모니터링 오류)

- 적절한 로깅과 모니터링을 통한 공격 감지 및 대응

💡 **취약점 예시**
1. 로그인, 인증 실패, 권한 설정 등 중요 기능 수행에 대한 로깅이 없는 경우
2. 일정 주기로 로그에 대한 백업 절차가 없는 경우
3. 로깅 및 모니터링이 필요한 부분을 명확하게 구분해서 로깅하지 않아, 불명확한 로깅 및 모니터링을 하는 경우

💡 **예방 방법**
1. 모든 로그인, 접근 제어, 인증 실패에 대해 로깅을 하고 정기적인 백업을 통해 보관
2. 로그 관리 솔루션 등을 활용하기 위해 적절한 형식으로 로깅이 생성되는지 확인
3. 의심스러운 활동을 감지하고 신속하게 대응할 수 있도록 임계치를 설정하고 모니터링
4. 침해 사고 대응 및 복구 계획 수립

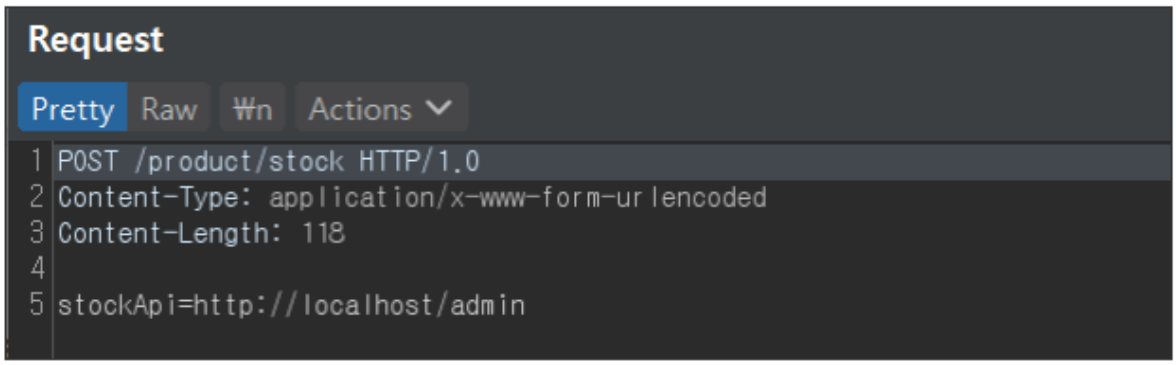
> 스캐닝 방법 : 로그인 및 접근에 대한 적절한 로깅기록 확인 후 양호 미흡 체크
> 로그인 실패 기록 및 실질적 보안관제사가 모니터링을 확인하는 것도 물리적인 부분이므로 확인이 어렵다. **<구현불가>**

A10: Server-Side Request Forgery(SSRF, 서버 측 요청 변조)

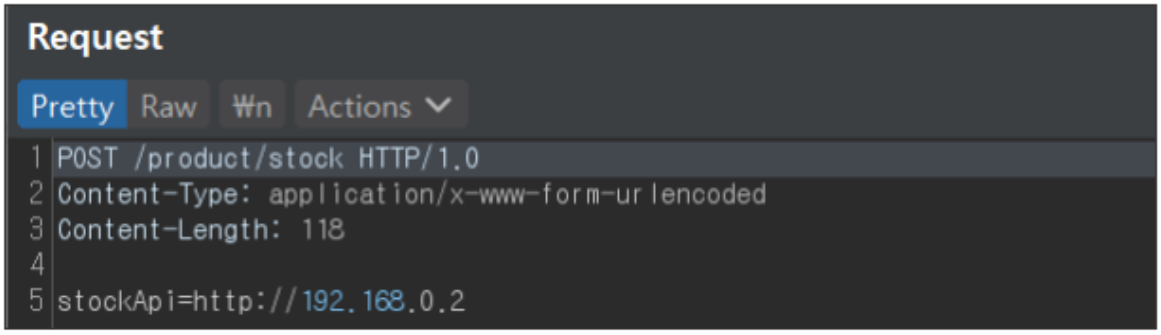
- CSRF보다 더 위험한 공격
- 서버 측에서 위조된 요청을 보내도록 하는 취약점

💡 **취약점 예시**
서버가 적절한 검증 절차 없이 사용자 요청을 로컬 혹은 원격 리소스에 접근하도록 하는 경우

💡 **예방 방법**
1. 서버가 속한 내부 네트워크끼리 통신할 때에도 방화벽을 통해 접근제어 규칙을 적용
2. 모든 사용자 제공 데이터에 대한 검증
3. 사용자 요청에 대한 서버 측 수행 결과 검증



로컬 서버에 대한 공격



원격지에 대한 공격

> 스캐닝 방법 : 방화벽 접근제어 규칙이 적용되어 있는지 체크